Trading off Accuracy, Timeliness, and Uplink Usage in Online GPS Tracking

A.B.M. Musa, James Biagioni, and Jakob Eriksson

Abstract—In an online GPS tracking system, a fundamental trade-off exists between *timeliness*, the average time interval between a recorded change in location and a change in the reported location; *accuracy*, the average error between the actual location and the reported location; and *uplink usage*, the average amount of data used per second of tracking.
While tracking efficiency has been addressed in the literature, our *thrifty tracking* system presents the first unified view of timeliness, accuracy, and uplink usage, allowing the user to specify desired targets for any two of these objectives, while optimizing the third. We also provide a closed-form characterization of this three-way trade-off, and demonstrate that our system converges to the predicted performance.

Index Terms—Spatial databases and GIS, Mobile communication systems, Mobile Computing, Client/server, Data compaction and compression, Data communications, Data communications aspects, System applications and experience

1 Introduction

Tracking assets and people using the global positioning system is becoming increasingly popular. Applications of GPS tracking are widespread, including anti-theft lowjack devices, freight logistics, and public transit arrival time prediction, as well as widespread crowd-sourced traffic data collection using smartphone apps.

The energy consumption of GPS devices has been the subject of much research in the past several years [2], [3], [4], [5], [6]. In GPS tracking applications, however, energy is arguably a secondary concern: typically energy is either abundant, as in most vehicular applications, or already expended, as in the crowd-sourced traffic maps example above. Any energy savings strategy used in an existing system would be applied before the techniques presented here.

Beyond energy consumption, a fundamental tradeoff exists in an online GPS tracking system, between *timeliness*, the average time interval between a recorded change in location and a change in the reported location; *accuracy*, the average error between the actual location and the reported location; and *uplink usage*, the average amount of data used per second of tracking. For example, we may achieve timeliness and accuracy close to the capability of the GPS receiver itself, but only at significant uplink usage, or excellent accuracy at virtually zero uplink usage using a offline data collection approach, typically imposing unacceptable delays. At the typical 1 Hz GPS update frequency, a few bytes worth of location update seems a mere pittance compared to the bandwidth hogging ways of modern smartphone users. However, including mandatory overhead in uplink usage, the above can adds up to over 200 MB/month, as we show in §4.

The primary contributions of this paper are: (a) a characterization of the inherent trade-off between the conflicting objectives $accuracy$, $timeliness$, and $uplink\ usage$, (b) a characterization of tracking behaviors currently used in practice, based on large-scale GPS probe data, (c) a unified sampling framework that allows the user to specify performance targets for two out of three parameters: *budget*, *error*, or *delay*, while it optimizes the third, (d) a unified extrapolation method that predicts future movements based on current conditions, (e) an end-to-end evaluation of the above methods on a large and diverse collection of GPS traces, and (f) public access to the source code and datasets [7].

2 State of the Practice

To gain an understanding of typical online GPS tracking behavior we studied a dataset consisting of 1.6 billion GPS points, collected by 25 different data providers, over the period Aug 2010–Aug 2012. For privacy reasons, individual traces were split into short, de-identified and disconnected *probes* consisting of a smaller (and highly variable) number of GPS points, before we received them. This means we cannot use this dataset for evaluation purposes (for evaluation we used datasets at Table 2), or to study individual device behaviors in detail. However, we can gain a good statistical picture by studying the sampling behavior exhibited within each individual probe. Figure 1 shows a histogram of time intervals between samples, across all probes. A clear pattern of periodic reporting emerges, with tell-tale peaks at periods 1, 5, 15, 30, 60, 90, 120, 180, 240 and 300 seconds. After removing these clearly periodic samples, 11.4% remain.

- *A. Musa (amusa2@uic.edu), J. Biagioni (jbiagi1@uic.edu) and J. Eriksson (jakob@uic.edu) are with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL, 60607.*
- *An earlier version of portions of this work previously appeared as a short-paper at ACM SIGSPATIAL GIS 2013 [1].*

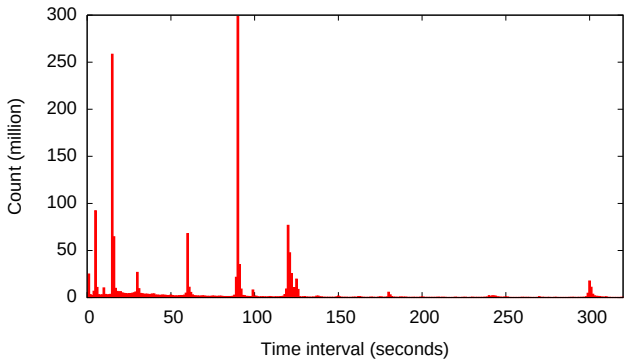

Fig. 1. Histogram of intervals between ∼1.6B location reports from 25 providers, illustrating the periodic nature of contemporary GPS tracking.

A majority of non-periodic transmissions that we sampled from Figure 1 coincided with likely stops, Controller Area Network (CAN) bus events such as ignition on/off events, etc. By the simple histogram analysis used above, one intermittent sample can make the interval to a succeeding otherwise periodic sample appear non-periodic. Thus, the 11.4% reported above is a likely overestimate of the proportion of non-periodic transmissions by up to $\approx 2\times$. Throughout, we were unable to find any evidence of spatial sampling (i.e., every so many meters), speed or bearing change-based sampling, or even policies as simple as not transmitting when stationary: all providers show several back-to-back transmissions with identical locations.

Some GPS tracking systems use additional sensors (e.g. accelerometer) to assist in duty-cycling the GPS receiver. Use of such sensors for informing sampling in addition to duty-cycling is conceivable, but somewhat unlikely: if the GPS receiver is active, the GPS trace itself contains all the information needed to make a sampling decision. Conversely, if it is inactive, no sampling decisions need to be made.

Thus, both anecdotally and based on this limited quantitative study, we believe there is ample room for improvement to the current state of the practice in online tracking.

3 GPS Tracking Literature Review

Due to the high power consumption of GPS receivers and their popular use in mobile, energy-limited devices, many researchers have focused on improving the energy efficiency of GPS tracking [2], [3], [4], [5], [6], [8], [9], [10]. By contrast, we assume that power is plentiful, or that the GPS is already active for a primary application. GPS trace compression [9], [11], [12], [13], [14], [15], [16], [17] is an often-used approach for producing a compact representation of a trace, however, while these techniques can be helpful in reducing the size of each transmission, they cannot reduce the number of transmissions without sacrificing timeliness.

Although we do make use of existing GPS compression techniques, the focus of our work is online tracking, where forwarding decisions are made as GPS points become available. The seminal work in this area was published by Wolfson and Sistla et al. [18], [19], [20], [21], where they present methods for updating databases that track moving objects. Similar to our work is [21], where they propose update policies based on dead-reckoning, wherein the moving object and database maintain a synchronized notion of the object's position between updates, and update the object's database position only when the distance between the object's actual and expected location deviate by too large of a margin. Our work builds upon theirs in two ways: (1) whereas their system requires knowledge of the object's expected path of travel in order to perform extrapolation, our system does not impose any such requirement, instead extrapolating the object's future path of travel using techniques that only rely on its history of previous locations. (2) while their system permits tuning of the update cost coefficients (used to control the trade-off between factors in optimizing performance), our system permits direct control over the costs themselves, giving users the ability to set hard limits on their range of values, and enabling concrete performance guarantees.

More recently, work by Čivilis and Jensen et al. [22], [23], [24] builds directly on the efforts of Wolfson and Sistla et al., and develops an architecture that lays the foundation for our work by adding support for several alternate extrapolation techniques. Our work extends upon theirs in two specific ways: (1) whereas their system requires the *manual* selection of one particular extrapolation method, our system provides a unified extrapolator that *automatically* selects the best method for the current conditions. (2) while their system controls the transmission of location updates through the specification of a maximum error-threshold, our system permits specifying two out of three performance criteria (error, budget, or delay), while it automatically optimizes the other.

Existing work by Lange et al. [25], [26], [27] looks at the problem of *online* trajectory reduction, where the objective is to store an approximation of a moving object's trajectory with the fewest possible vertices, while simultaneously ensuring it doesn't deviate from the actual trajectory by more than some specified accuracy bound. While this work bears some similarity to ours, it differs in two ways: (1) the authors' primary concern is storing a compact and accurate trajectory in real-time, with object tracking being only a secondary concern. Conversely, we're primarily concerned with object tracking, with accurate trajectory representation and storage being only an incidental concern. (2) the technique their system uses for extrapolation is limited to simple linear dead-reckoning, whereas we provide myriad alternatives through the use of our unified extrapolator.

Protocol	Data Usage per Update	Data Usage per Day
UDP	84 bytes	7 MB
TCP session (cont.)	168 bytes	14 MB
TCP session (sep.)	840 bytes	69 MB
HTTP / REST	1218 bytes	100 MB

TABLE 1
Data usage billed for 4 protocol choices, using 8 byte payload, 1 second mean interval, operating 24 hours.

4 Mobile "Data Usage" Billing

Reducing data consumption for online tracking is an important goal. To understand how the data usage is measured and how much overhead is imposed, we conducted a set of experiments on the AT&T wireless network. Overall, we found that for frequent small-size transmissions, such as from GPS, the data usage including overhead adds up quickly. Below, we describe this in more detail.

4.1 Experiments with AT&T Wireless

Typical off-the-shelf GPS receivers produce GPS readings at 1 Hz. Readings always contain a (time, latitude, longitude) tuple, but may also include speed and heading information, altitude and various signal quality measures. Thus, the size of a GPS location update to be transmitted over the cellular modem may be in the 8–100 byte range.

As an initial test, we programmed several phones to send a 1 byte UDP packet every second over their 3G uplinks, and monitored the resulting "data usage" as reported by their cellular data plan provider. Over the course of a day, the 86,400 individual packets observed by our receiving server (carrying 86,400 bytes of data payload) incurred a total "data usage" of 6.9MB, 84× the actual data sent. Further experiments, with various protocols, payload sizes and transmission intervals confirm this behavior.

For Table 1, which presents a brief summary, we used a truly bare-bones 8-byte update payload sent at 1 Hz, and determined the actual data usage charged. Here, TCP (cont.) represents a single continuous TCP session, whereas TCP (sep.) establishes a new session for each update. Anecdotally, we have seen both methods employed by commercial tracking devices. The HTTP method, typical for so-called RESTful web services, uses the built-in URLConnection class in iOS to upload a location update, closing the TCP session between updates. We would expect HTTP data usage to fall between TCP (cont.) and TCP (sep.) for servers supporting HTTP keep-alive.

Judging from our experiments, the model for data usage billing includes all headers, and rounds up to the nearest 42-byte increment. In general, data usage billing can be calculated as follows:

$$data\_usage = \sum_{p \in P} 42 \left\lceil \frac{size(p) + c}{42} \right\rceil$$

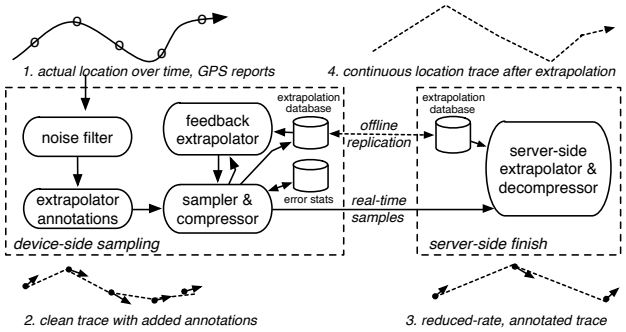

Fig. 2. Architectural diagram of a thrifty tracking system. Adaptive sampling techniques rely on feedback from an extrapolator executing on the device, mirroring the extrapolation done on the server.

where P is the set of all packets transmitted, uplink and down-link, network layer and up. $size(p)$ is the size of packet p network layer and up, and $c \geq 10$ bytes represents link-layer overhead. Our preliminary experiments indicate $c \geq 10$ bytes. One implication of the experiments above is that on the AT&T network any transmission will incur a minimum of 84 bytes of "data usage". Other networks may differ, but the more general intuition here is that every individual transmission has substantial overhead, and network providers are very likely to pass on this overhead to the user in the form of data usage.

Overall, beyond the rather obvious step of switching from HTTP to UDP or continuous TCP for transport, reducing the number of transmissions is the best way to reduce the "data usage" of GPS tracking. By carefully choosing what and when to transmit, significant improvements can be achieved in data usage, accuracy and/or timeliness.

5 Thrifty Tracking Overview

Figure 2 illustrates our general thrifty tracking architecture which accommodates the existing tracking methods known to us [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], as well as a variety of new tracking methods introduced in this paper. Starting in the top-left of the figure, a GPS receiver samples the (continuous) device location, often with a frequency of 1 Hz or higher. If a GPS energy conservation mechanism is in use, this is done before the thrifty tracking system receives the GPS samples. The incoming "raw" GPS trace (1) is first passed through a filter to remove any obvious outliers. An *annotator* then decorates each point with additional information not provided by the GPS receiver. This could include quantitative estimates of acceleration or angular velocity, as well as calculated speed, heading, current road segment, etc.

Next, the decorated GPS trace (2) is passed to a sampler. The sampler unilaterally decides whether to

	OSM	MSMLS	UIC Shuttle
Individuals	5000+	17	-
# GPS Points	12.4M	500K	1.9M
Recording Int.	1 s	10 s	1 s
Total Duration	3450 h	1490 h	530 h
Geog. Dist.	Moscow	Seattle	UIC
Type	city & highw.	suburb & highw.	city

TABLE 2
Experimental data summary. All UIC Shuttle buses treated as a single individual.

forward a given trace point, with any necessary annotations, to the server. This decision is made based on one or more factors such as time, changes in speed, heading, recent data usage history, data usage budget or target, battery level, and perhaps most importantly—extrapolation error, as discussed next.

The resulting sampled trace (3) is then fed to two identical extrapolators: one running on the server, and one running on the mobile device. An extrapolator takes a sampled trace and an extrapolation database as input, and produces a *continuous* location estimate at the current time.[1] Here, the extrapolation database is a replica of a database on the server side, containing any trained parameters and/or map data needed for each extrapolation method in use. On the server side, the extrapolator output (4) is made available for use by the trace consumer. On the client side, an identical extrapolator produces a continuous location estimate for local use in computing the current extrapolation error. This is provided as feedback to *error-aware* samplers. By comparing the output of the local extrapolator against the incoming raw GPS location, an error-aware sampler makes its forwarding decision based on the difference between the current estimate and the location reported by the GPS.

Note that except where otherwise noted, the term *error* refers to sampling error, as incurred by not transmitting every GPS coordinate. This error is in addition to the positional error in the coordinates produced by the GPS receiver. Thus, for our purposes, the original GPS stream is the "ground truth".

The primary parts of this system are the extrapolator (§6) and the sampler (§7), which are described in more detail below. In §6, we first discuss several free-space and map-based extrapolation techniques, and then describe our unified extrapolator, together with a comparative evaluation. In §7 we briefly discuss basic sampling techniques, followed by our adaptive samplers which outperform the basic samplers while allowing the user to specify high-level performance targets, rather than technical parameters. Finally, in §8 we evaluate the combined thrifty tracker, consisting of our unified extrapolator and sampling framework.

Our evaluation of these techniques is based on three distinct datasets, summarized in Table 2. The OpenStreetMap [28] ("OSM") data was scraped from the OSM website, but originally donated by volunteers from Moscow, Russia, and its outlying areas, the Microsoft dataset ("MSMLS") was collected by Microsoft employee volunteers [29], and the UIC dataset was collected from campus shuttle buses instrumented by the authors. In total, our evaluation spans over 5000 individuals, 14.8 million data points, and almost 5500 hours of recorded locations. For the OSM dataset, we used 1/4th of the data for training and the remainder for evaluation. For the smaller UIC and MSMLS datasets where data is somewhat scarce, we used 5-fold cross-validation.

Due to space constraints, our discussion and evaluation of results in Sections 6–7 focuses on the OSM dataset, which represents our largest and most diverse collection of location samples. The results are similar for other datasets, and we show the end-to-end performance across all datasets in Table 3 and Table 4. Our evaluation on this diverse dataset (city, suburb, highway) shows that our system generalizes well across various scenarios.

6 GPS Trace Extrapolation

By predicting the future location of a device, or *extrapolating* its location trace, improvements can be made to both the timeliness and accuracy of tracking. The most basic *free-space* extrapolation method ("Constant Location" (CL)) predicts that the future location, for all times in the future, will be the same as the most recently reported location. By applying this extrapolation method, we improve the timeliness of our tracking: we are now able to provide an immediate estimate, at any point in time. However, this gain in timeliness is matched by a loss in accuracy: for a moving device, predictions made by this extrapolator grow increasingly inaccurate with the time since the last update.

In this section, we explore how more sophisticated extrapolation methods described in prior work such as [19], [21], [23], [24] can be used to improve accuracy. Specifically, we evaluate the following three free-space techniques: Constant Velocity (CV), which produces a straight-line path from the most recent location, at the velocity from the most recent report, and Constant Acceleration (CA), which is based on CV but also includes estimated acceleration. We also introduce a minor variation, named Constant Deceleration (CD) which performs the same function as CA when acceleration is less than zero, otherwise it performs the same function at CV. Finally, we evaluate the use of advanced map-based extrapolators, described below. In §6.3, we propose a unified extrapolation technique that combines these free-space and map-based methods using a machine learning approach, to further improve extrapolation performance and therefore tracking efficiency.

6.1 Map-Based Extrapolation

If movements are restricted to a known map, or if past movement history is available, this information

1. This is not to be confused with an interpolator, which produces a continuous location estimate between past location reports.

may be used to improve extrapolation performance over the free-space methods above. Map-based techniques naturally rely on online map-matching. For this, we use a Viterbi map-matcher similar to those described in [5], [30]. These techniques are both batch-oriented, and output the maximum probability path at the end of trace processing. To make Viterbi map matching work for online purposes, we instead use the maximum probability edge at each time step, at the cost of producing less accurate results.

Our map-based extrapolator operates by traveling along the current road until an intersection is reached. Once at the intersection, it may either stop there (M0), continue straight through the intersection if possible (M1), or decide how and whether to turn based on past movement history (MM0/MM1).

While we in principle support map-based extrapolation anywhere in the world, any practical solution would retrieve only local map data (a serialized graph representing the road map and a lookup table for intersection turn probabilities) either at install-time, or at regular service intervals to keep the map up to date. This could be done using offline methods, or over a Wi-Fi link, so as to not incur "data usage" for occasional map data downloads. Using this approach, we expect that the most users would be well served by map-based extrapolators using tens of megabytes of storage for map data.

6.1.1 Trace-Driven Turn Prediction

If historical traces are available, these may be used to learn the most likely choice at each decision point as shown in [31]. This can be thought of as an n^{th}-order Markov model, where street segments are represented by states, and turns by transition probabilities. During extrapolation, when encountering an intersection the turn with the highest probability indicated by the model is taken and extrapolation continues along the new subsequent street. If the model contains no past movement history for a given intersection, we have a choice of either stopping at the intersection (MM0), or continuing straight through (MM1). For our evaluation we elected to use a 10^{th}-order Markov model, based on the good performance shown on this task in [31]. We use separate sets of past traces to train and test the model.

6.2 Individual Extrapolator Performance

Given the variability of GPS traces, it is unclear which of these extrapolators most accurately predicts future movements for an arbitrary trace. To evaluate the extrapolators described above, we compare their predicted locations to the measured locations throughout the three datasets described in §5. More specifically, for each trace L, and for each time i of the trace, we compute the distance $\delta^i(j) = |\hat{L}^i_j - L_j|$, between the extrapolated locations $\hat{L}^i_j$, and the corresponding actual locations L_j, $\forall j > i$. For each time i, we then produce two numbers: D^{max}_i, the number of seconds that elapse before $\delta^i(j) >$

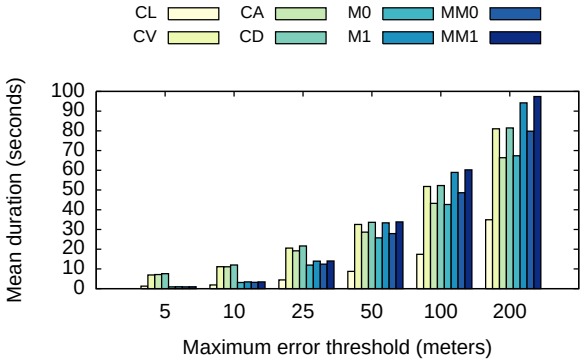

Fig. 3. $\overline{D^{max}}$ for eight extrapolators on our OSM dataset, for varying values of max (higher is better).

max, where max is a maximum tolerable distance error threshold, and

$$E^{\Delta t}_i = max(\{\delta^i(j) : j = i+1, ..., i+\Delta t\})$$

the maximum error incurred over an interval of Δt seconds following the location update. For each extrapolator we then report its mean duration

$$\overline{D^{max}} = \frac{1}{|L|}\sum_{i=0}^{|L|} D^{max}_i$$

over all traces, which indicates how long an extrapolator "lasts" given a maximum error threshold, and its mean maximum error

$$\overline{E^{\Delta t}} = \frac{1}{|L|}\sum_{i=0}^{|L|} E^{\Delta t}_i$$

over all traces, which describes how well it performs over a fixed time interval.

In this section, we report on extrapolator performance in isolation, using a historical dataset rather than a client-server deployment. For extrapolators that require training, we use separate test and training sets for evaluation, as described in §5. A similar technique would be used for a deployed system: the adaptive extrapolators would be bootstrapped using a representative dataset. This could potentially be updated offline, either using only the devices own collected data, or with recent global statistics, but we did not evaluate this. Due to the lack of adaptation, our results are likely to somewhat underestimate the long-term performance of a deployed system.

Figs 3 and 4 show the value of $\overline{D^{max}}$ and $\overline{E^{\Delta t}}$ for eight extrapolators on our OSM dataset, for varying values of max and Δt. Overall we observe that the free-space extrapolation methods outperform their map-based counterparts when the values of max and Δt are low. However, as we increase max and Δt the map-based methods eventually reach performance parity, and then outperform the free-space methods by an increasingly large margin.

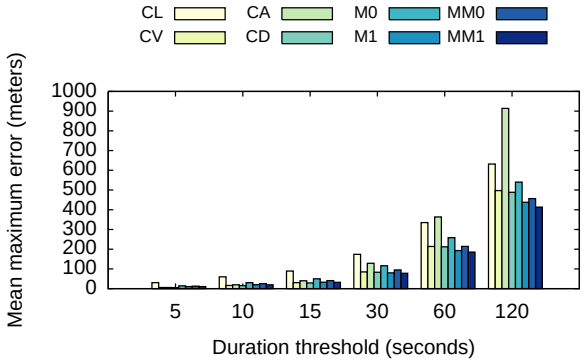

Fig. 4. $\overline{E^{\Delta t}}$ for eight extrapolators on our OSM dataset, for varying values of Δt (lower is better).

Within the collection of free-space extrapolators we find that CD performs best in all cases, followed closely by CV, then CA, and lastly CL. The superior performance of CD can be attributed to its harnessing the best aspects of CV and CA, while not inheriting their limitations. When constant velocity or acceleration is detected in the original trace, CD produces an extrapolated trace identical to that of CV. However, when deceleration is detected, CD reduces the velocity of the extrapolated trace at the detected rate of deceleration, until zero velocity is reached. CD's ability to replicate this real-world phenomena is key to its performance advantage. CV follows closely behind CD in performance, limited only by its inability to reduce an extrapolated subject's velocity. CA follows further behind, as its tendency to continuously *increase* velocity when acceleration is detected (up to 60 mph) is largely inconsistent with reality. Finally, the universally poor performance of CL is a direct result of our OSM data consisting of traces where the subject is often moving. Because CL is only able to predict the current location for all times in the future, this results in quickly growing extrapolation errors.

Within the collection of map-based methods we find that trace-driven turn prediction (MM0/MM1) performs better than un-trained techniques (M0/M1). This finding is a direct result of trace-driven turn prediction leveraging past behavior to predict the future. Because it doesn't have to stop at intersections, it is able to extrapolate further into the future than M0, and since it has knowledge of past turns at intersections, it has a better idea of which direction to follow than M1. We also observe that given the option of either stopping or continuing straight through an intersection, in both the trained (MM0/MM1) and un-trained (M0/M1) cases, continuing straight through always results in better extrapolation performance. This is a largely intuitive result, as people traveling along roads tend to proceed straight through intersections far more often than they turn (approximately 90% of the time in our datasets), making an extrapolation of "straight-ahead" a reasonable prediction in the absence of any additional information. Because M1 and MM1 consistently outperform M0 and MM0 (respectively), we only present results from M1 and MM1 going forward.

6.2.1 Overall performance analysis

Our finding that map-based extrapolation methods fail to work well at low error-thresholds is seemingly counterintuitive, as they have far more information with which to predict the path of future travel. However, there are at least two factors that limit their accuracy in situations where strict error tolerance is desired: *road position* and *GPS error*.

Because digital maps are a simplification of reality, often representing multi-lane roads by simple bidirectional paths, they are unable to represent the position of a subject in a way that accurately reflects their real-world location. Instead, they have to "snap" the subject to their closest likely road position, often introducing one or two lane-widths of error before extrapolation along the map even begins. Moreover, because the extrapolator is tasked with replicating our real-world location traces, its performance suffers further as a result of the measurement error in our recorded locations. Commodity GPS receivers typically report locations with 5–10 meters of error under ideal conditions, and in the presence of tall buildings, loss of signal and multi-path effects occasionally produce errors in excess of 100 meters. Because these errors are a phenomenon our map-based extrapolators are unable to predict and reproduce, when they occur in our traces and deviate from the road, they will be reported as extrapolation errors.

From this we conclude that free-space extrapolation techniques thrive in low error-bound conditions because they begin their extrapolation from the measured GPS location directly, and proceed without their predicted path having to follow an inflexible model. It is worth noting here that while the vehicle may well be located on the street, much as a map-based extrapolator would suggest, our evaluation is based on reproducing the original GPS trace, not estimating the "true location" of the vehicle.

6.3 Unified Extrapolation

Based on our observations in §6.2 we conclude that while certain extrapolators work well under certain conditions, no single extrapolator offers the best performance in all circumstances. To address this, we propose a unified extrapolator which automatically selects the best extrapolation method for the current circumstances.

Since our unified extrapolator cannot know what will happen in the future, this is a challenging task with no guaranteed results. We cast this as a classification problem, relying on past experience to train a classifier: supervised learning applies here as the recorded history reveals exactly which extrapolator works best.

Given that CD and MM1 are the best performing extrapolators on our OSM dataset for low and high (respectively) error and duration thresholds, our first

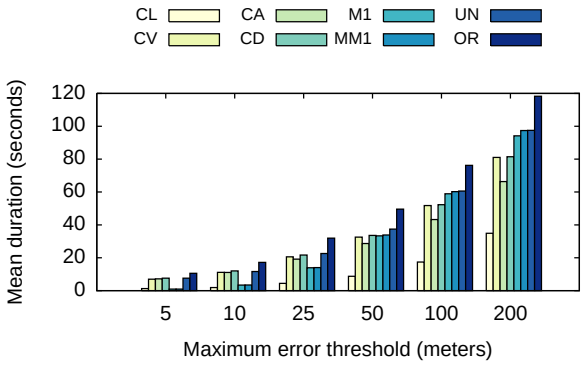

Fig. 5. $\overline{D^{max}}$ for eight extrapolators on our OSM dataset, for varying values of max (higher is better).

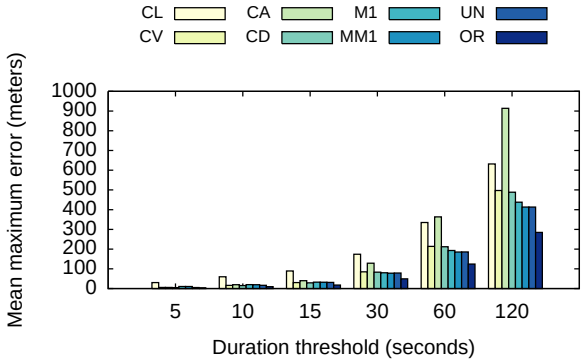

Fig. 6. $\overline{E^{\Delta t}}$ for eight extrapolators on our OSM dataset, for varying values of Δt (lower is better).

approach was to build a simple two-class decision tree that selected either the CD or MM1 extrapolator. While this classifier performed well in practice on our OSM dataset, we found that it did not generalize well to our MSMLS and UIC datasets, where other extrapolators (CL, CA) outperform CD and MM1 over the same range of error and duration thresholds.

Therefore, in order to make our classifier more general-purpose we adopted a multi-stage classification approach. Our experiments with the simple CD/MM1 classifier taught us an important lesson: making a classification decision between the free-space and map-based extrapolators is very practically useful, as there is a distinct "cross-over point" along the threshold ranges where the performance advantage between these two classes of methods is exchanged. With this in mind, we constructed the first stage of our multi-stage classifier as a decision tree trained to select between the free-space and map-based extrapolators. Then, to ensure the generality of our approach, we separately trained two additional decision trees: one to select amongst the free-space extrapolators, and one to select amongst the map-based extrapolators. Putting it all together, our multi-stage classifier then operates as follows: input samples are first presented to our first-stage decision tree in order to determine the appropriate free-space/map-based extrapolator class. Then, based on that decision they are either passed to the second-stage free-space extrapolator decision tree, or second-stage map-based extrapolator decision tree, in order to determine the specific extrapolation method to use. For all of our decision trees we used the implementation provided by the Scikit-learn [32] machine learning library.

We train all of our decision trees using the following set of features, drawn from a 60-second sliding window of locations immediately preceding the current sample: distance between the current sample and its map-matched edge, previous samples' mean speed, difference between the current sample's speed and previous samples' mean speed, difference between the current sample's acceleration and previous samples' mean acceleration, current sample's distance to the previous samples' mean location, and the current sample's distance to the immediately preceding sample. These features were identified among a larger collection using a Tree-based estimator [33] to compute their classification importance, and determined to be significant factors in selecting the best extrapolation method. The storage requirement for the trained model is tens of kilobytes.

6.3.1 Unified extrapolator performance evaluation

To determine a loose upper bound on the performance of our unified extrapolator, we use an "Oracle" extrapolator that uses offline access to our recorded GPS traces to choose the best extrapolation method for any given sample. The Oracle extrapolator is used for comparison purposes only. Because it bases its decisions on knowledge of "future" events, it is not feasible for use in a real implementation. However, it provides a valuable comparison point for evaluation. The performance of the Oracle extrapolator is shown as the column labeled *OR* in Figs 5 and 6. In the worst case we can see that the Oracle's performance matches that of the single-best extrapolation method, and in the best case far exceeds the performance of any alternative, suggesting that dynamically selecting which extrapolator to use can potentially result in significant performance gains.

As Figs 5 and 6 show, the Unified extrapolator (column labeled *UN*) meets the performance of the best individual extrapolator for any given value of max or Δt, and in some ranges exceeds the performance of all individual extrapolators by adaptively selecting the best method to use at any particular moment in time. Generally speaking, the Unified extrapolator performs robustly across the full range of max and Δt values, achieving our goal of creating a *single* extrapolation method that flexibly adapts to its circumstances. The remaining disparity between the *UN* and *OR* columns suggest there may still be room for improvement in adaptively selecting the best extrapolation method, but this is a challenge we leave for future work.

7 Adaptive Sampling with Extrapolator Feedback

Online tracking systems trade off three performance attributes: $accuracy$, $timeliness$, and $uplink\ usage$. More formally, we have empirically found that their relationship takes the form

$$\mu = A * \frac{1}{\delta^B \tau^C} + D \quad (1)$$

where μ is the mean data usage in bytes per second, δ is the mean error experienced vs. the original trace in meters, and τ is the mean delay incurred in seconds. For $\delta < 10$ meters and $\tau \geq 4$ seconds, $B = 0.5$ and $C = 0.75$ is an excellent fit with the performance of all three samplers described below[2].

Note that while this equation describes the expected performance, and provides a useful benchmark to strive toward, it does not describe how to achieve it. In other words, what sampling policy achieves the expected δ, if both τ and μ are given? This is described in §7.3–7.5.

For delays below 4 seconds, Eq. 1 still applies, though with different exponents and coefficients. This is in part due to the minimum 84 byte transmission size in the cellular network we use for evaluation. This leaves room in each packet for up to four samples at no additional uplink usage. For delays below 4 seconds, we find $B = 0.75$ and $C = 0.25$ provide an excellent fit with our results. For $\delta < 10$ meters, the effects of GPS error dominate the sampler performance, rendering the results incompatible with Eq. 1.

7.1 Adaptive vs. Periodic Sampling

Uniform periodic sampling with a period of p seconds is popular today (see §2). Uniform sampling implies

$$\mu = \frac{M}{p},$$

where M is the data uplink usage of the minimum size transmission, or 84 bytes by our measurements (§4). In such systems, the location reported to the user typically remains unchanged between location updates. Within our framework, this is captured by constant location (CL) extrapolation, and a sampling delay τ of zero seconds.

Intuitively, while periodic sampling with no delay provides highly predictable data usage, it provides very loose guarantees on accuracy (bounded only by the maximum speed of travel). Alternatively, sampling at a fixed distance interval provides a strict error bound, but only loose constraints on data usage. Finally, aggregating several samples into each periodic transmission improves accuracy and reduces data usage over sending a single sample, at the cost of introducing delay.

Replacing CL extrapolation with the unified extrapolator from §6 may reduce the error of a periodic sampling policy by 2–4× (see Fig 6). However, further efficiency gains can be had by adapting the sampling (transmission) policy to current conditions. With our system, the user specifies target values for two of the three variables: *accuracy*, *timeliness*, *uplink usage*, and the system produces an online sampling policy that optimizes the third. Specifically, the user specifies *accuracy* in terms of max error (in meters), *timeliness* in terms of fixed delay (in seconds), and *uplink usage* in terms of uplink budget (in average bytes/sec). Thus, the user may specify an uplink budget of 4 bytes per second, and a fixed delay of 30 seconds, and our system will produce a sampling policy that minimizes the resulting mean error.

The specified maximum error bound and fixed delay are enforced by our system at all times. On the other hand, the specified budget in average bytes/sec is a long term goal. However, for all practical purposes, the budget can be considered as instantaneous, since our system aims to enforce this throughout the sampling process (described more in §7.4).

Internally, this is implemented in the form of three different samplers—one for each parameter to be optimized: mean uplink usage (§7.3), mean error (§7.4), and mean delay (§7.5). If necessary, these samplers are easily extended to support additional variations (e.g. accounting for the connection setup cost in addition to the regular transmission cost, free data after 9 pm etc.).

In §7.6 we show that although these samplers follow different optimization strategies, the expected performance of each sampler falls on or near a shared solution surface that conforms to Eq. 1, validating the effectiveness of their respective strategies. We also compare the performance of adaptive samplers to the corresponding periodic (straw-man) samplers in §7.3-§7.4.

7.2 Adaptive Sampling Overview

By estimating the location of a device without additional transmissions, extrapolation naturally helps improve accuracy and timeliness at the server. However, extrapolation can also be used to improve efficiency at the sender. Here, the sender replicates the extrapolation process performed at the receiver, enabling it to directly observe the extrapolation error incurred at the server. This, in turn allows the sender to choose the samples it transmits to maximize the accuracy gained from each transmission.

For maximum timeliness, a sampler must decide whether or not to transmit each sample as soon as it arrives, allowing relatively little room for maximizing sampling efficiency. However, if the user is willing to tolerate a fixed delay in the reporting, an additional performance gain can be achieved by choosing when to transmit a sample. Because of the delay, the sampler can essentially look ahead and see if significant extrapolation errors will occur because of not transmitting some samples. Thus by transmitting such samples, the sampler can mitigate the high errors before they occur. Fig 7 shows an example of this. Here, the solid disks

2. With A=65 and D=0.228

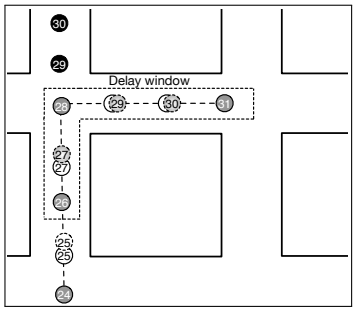

Fig. 7. By introducing a fixed delay, a sampler can send multiple samples in one packet, and may choose more effectively what samples to send.

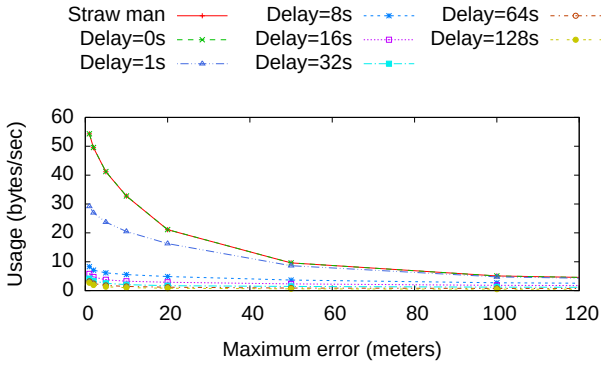

Fig. 8. Uplink data usage with increasing maximum error bound, for usage optimizing sampler with various configured delays.

(white or gray) are actual locations, the dotted white disks are extrapolated locations, the black disks are extrapolated locations (not transmitted), and the gray disks are transmitted locations. In this example, The subject takes a right turn in the intersection. Here, significant extrapolation errors (black disks) are made before the turn because of failure to predict the turn, but with the added delay the sampler can capture this and avoid the extrapolation errors. In addition, the sampler can apply GPS compression and transmit only a few key locations (gray disks). Then the server fills up for non-transmitted locations by interpolation (dotted disks with radial gray).

GPS compression [14] can yield substantial data usage savings with minimal accuracy loss, particularly for long delays. Here, samples are selected in the order of maximum error reduction until the errors for all original samples in the reconstructed trajectory fall below the given error bound.

7.3 Sampling for uplink usage optimization

With a maximum error bound and fixed delay configured by the user, the task of the sampler is to minimize mean data usage while enforcing the maximum error bound. For each incoming location from the GPS, the sampler measures the distance between the extrapolated trace and the current location. If the distance exceeds the maximum error bound, this sample must be transmitted. If zero delay is configured, the sample is transmitted immediately, updating the server and restarting the extrapolation with the new parameters.

With a non-zero delay of T seconds, the sample (and the surrounding window of samples) may be transmitted at any time between the present time and T seconds into the future. The optimal time to transmit is the one that minimizes the resulting error. Since the future is unknown, we use the mean maximum error from Fig 6 as a proxy for future errors.

For each time-step (i.e. second), we decide whether to transmit the current window containing the oldest sample (which will be dropped next) or defer transmitting in the hopes of finding larger errors (to be corrected) in the future. Thus, the current window should be sent only if it has a greater mean error than the expected mean error of all other candidate transmission windows. Whenever this condition holds, we transmit the window using our error bound GPS compressor.

Fig 8 shows the data usage as the maximum error threshold is varied, for different delays. Here, we use the Constant Location (CL) extrapolator, to highlight the behavior of the sampler in isolation. In §8, we evaluate the performance of the combined sampling framework and unified extrapolator.

The straw-man solution in Fig 8 transmits a single sample with a fixed distance interval equal to the maximum error threshold, thus guaranteeing that the error never goes beyond the user provided error threshold. With no delay configured, and using the CL extrapolator, the uplink usage optimizing sampler reduces to the straw-man, explaining their identical results.

While the straw-man provides a guaranteed maximum error bound, it does so at high uplink usage. However, as one might expect, this data usage decreases with increasing maximum error tolerance. Uplink usage optimizing sampling outperforms the straw-man by a considerable margin as soon as some amount of delay is configured, even as small as one second. Moreover, further gains are available when uplink usage optimizing sampling is used in combination with a more sophisticated extrapolator (see §8.1).

Fig 9 shows how the data usage decreases rapidly as the configured delay is increased, demonstrating the combined effect of adaptive sampling and compression at play.

7.4 Sampling for error minimization

The error minimizing sampler accepts a target uplink budget, B_{long} in bytes/sec, and fixed delay, d in seconds from the user. It then minimizes the mean error while meeting the uplink budget and delay targets. Intuitively, the larger the uplink budget, and/or the larger the delay, the smaller the mean error.

In contrast with the budget optimizing sampler, which enforces a strict error bound on each sample, the error minimizing sampler enforces a long-term uplink budget.

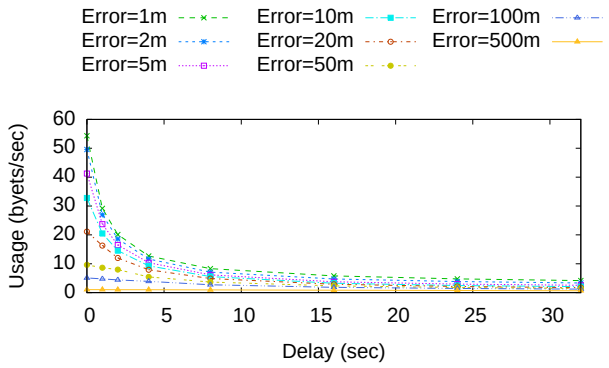

Fig. 9. Uplink data usage with increasing delay, for usage optimizing sampler with various maximum error bounds.

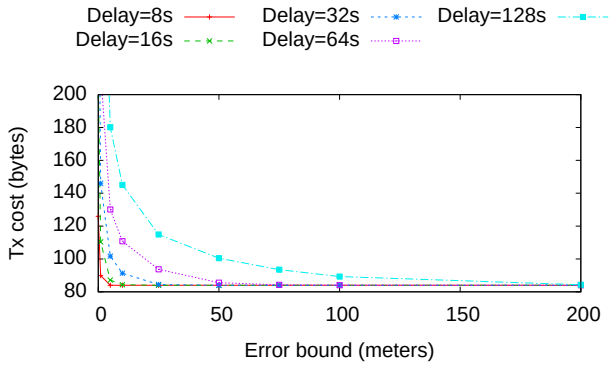

Fig. 10. Tx (transmission) cost with increasing error bound, for GPS compression with various delays.

The sampler's job, thus, is to decide how to best use the specified budget in the short-term, so as to minimize mean error while meeting the long-term budget.

Transmission decisions are initially based on the expected error for a given uplink budget B_{long} and delay d. Whenever a window exceeds the expected error, it is transmitted.

The expected error is a combination of the extrapolation error before the transmission window, and the compression error within the window. Fig 6 illustrates the extrapolation error, which grows with increasing duration. Fig 10, in turn, illustrates the compressor performance; here the transmission cost decreases with increasing error bound.

The expected uplink usage is a function of the expected extrapolation duration $t(e)$, the configured delay d, and the expected transmission cost $c_d(e)$ for the window that is eventually transmitted, or

$$\frac{c_d(e)}{t(e)+d} \tag{2}$$

Thus, to meet the specified budget B_{long}, we select the expected error e that minimizes $\left|\frac{c_d(e)}{t(e)+d} - B_{long}\right|$.

Although the expected error is a good threshold to begin with, current conditions can vary significantly from the expectation. In order to take actual, short-term

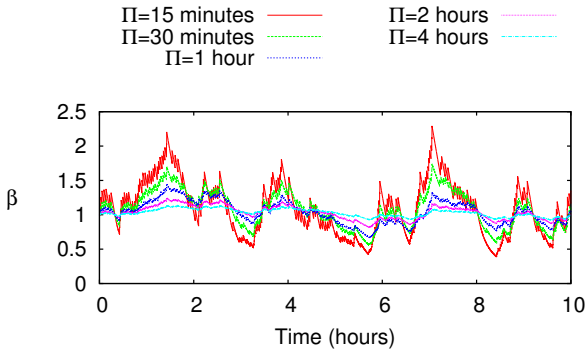

Fig. 11. β over time for error minimizing sampler with various Π (B_{long}=0.5 bytes/sec, delay=16s).

usage into account, we maintain the current $balance$ (in bytes), which is the difference between the uplink budget accrued over the sampler runtime up to the current time, and the total uplink usage incurred to this point. Usage is incurred whenever we transmit a window, and the balance builds at the user-specified rate of B_{long} bytes/sec.

Intuitively, if a large positive balance has accrued, the sampler can afford to lower its error target. Conversely, if the recent events have led to a negative balance, the error target may need to be raised in order to erase the debt. Accordingly, our sampler will transmit iff:

$$extrapolation\_error > \beta * expected\_error \tag{3}$$

Here, β encourages greater spending for positive balances, and conservation for negative balances. More precisely,

$$\beta = \begin{cases} \frac{B_{long}\Pi}{B_{long}\Pi + balance} & \text{if } balance >= 0, \\ \frac{B_{long}\Pi - balance}{B_{long}\Pi} & \text{if } balance < 0, \end{cases} \tag{4}$$

where Π is the period over which we desire the balance to return to zero. Let us observe a few key points along the curve described by β. When balance is zero, β is 1, meaning we should transmit only when the current error exceeds the initial expected error. If the balance should grow to be equal to the entire budget for a period, β shrinks to 0.5, representing a reduction by half in the maximum error tolerated before a transmission is made. Conversely, should a debt equal to the entire budget over a period ($B_{long}\Pi$) be accrued, β will reach 2, resulting in a much more conservative transmission policy. Finally, when transmitting the window, the same error threshold of $\beta * expected\_error$ is used for GPS trace compression.

Fig 11 shows how β evolves over time, for the various periods Π used in Eq. 4. Here, a short period results in quick adjustments to the current conditions while a longer period results in slower and smoother adjustments.

Fig 12 shows the mean error incurred vs. mean data usage, for several specified delays. Here, the straw-man solution transmits with a fixed period that conforms to

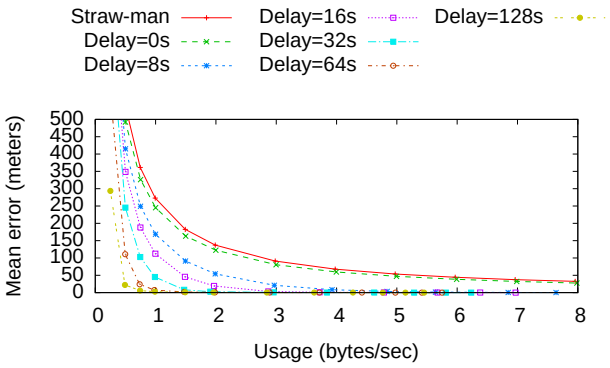

Fig. 12. Mean error with increasing usage, for error optimizing sampler with various configured delays.

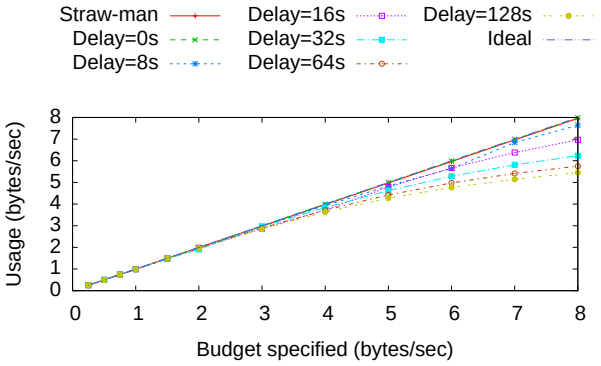

Fig. 13. Conformance with the specified uplink budget, for error optimizing sampler with various configured delays.

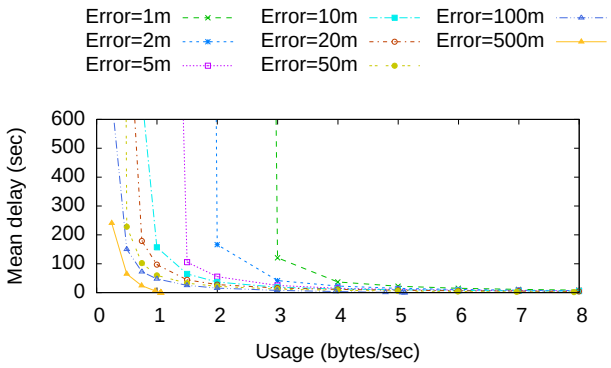

Fig. 14. Mean transmission delay with increasing usage, for delay optimizing sampler with various maximum error bounds.

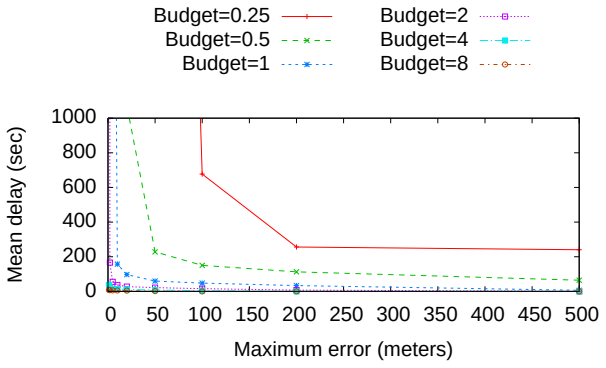

Fig. 15. Mean transmission delay with increasing maximum error bound, for delay optimizing sampler with various specified budgets.

the user-specified budget. Our initial advantage over the straw-man solution is substantially increased as we set a non-zero delay.

Fig 13 shows the budget conformance of the sampler, in terms of the incurred mean data usage vs. specified uplink budget, for several delay choices. While the sampler never exceeds the specified budget, we find that for generous budget/delay configurations, our sampler does not consume the entire budget. Since any significant error is already eliminated, the sampler instead accumulates a balance, to be spent if conditions should become adverse later on. Note that we show data usage, not given budget in Fig 12.

7.5 Sampling for delay optimization

Finally, our user may have an uplink budget and maximum error target in mind, but remain flexible in terms of the reporting delay. According to Eq. 1, increasing the delay reduces data usage. Our delay-optimizing sampler leverages this relationship to meet both uplink budget and maximum error targets simultaneously.

When sampling with a fixed uplink budget and maximum error, the uplink budget is a long-term goal, but the maximum error needs to be met for every location delivered to the user. Accordingly, we use the budget optimizing sampler described above (§7.3) internally, with the user-configured error target. We then dynamically adjust the delay to meet the configured budget in the long term.

The initial delay is based on the expected usage of the budget optimizing sampler, as per Fig 9. This estimate is calculated from the past movement history of the tracked device, and provides an adequate starting point. However, history is not necessarily a good indication of future performance, potentially resulting either in an unnecessarily long delay, or in usage well above the target budget.

To address this, we use the balance-based adjustment factor β introduced in Eq. 4. Here, given a long-term budget target B_{long}, the delay at each time step is selected using the balance-modified budget B_{long}/β. Recall that $\beta < 1$ for positive balances, and $\beta > 1$ for negative balances, with the exact value depending on the configured averaging period Π. Therefore, the delay-optimizing sampler selects the delay based on a larger budget value if a positive balance exists, and vice-versa.

Fig 14 shows how the delay varies with the mean uplink usage for several specified errors. As expected, delay decreases rapidly as more uplink usage is allowed. Similarly, Fig 15 shows how delay is reduced with increasing maximum error tolerance. Intuitively, the budget lasts longer with a lower error bound, meaning

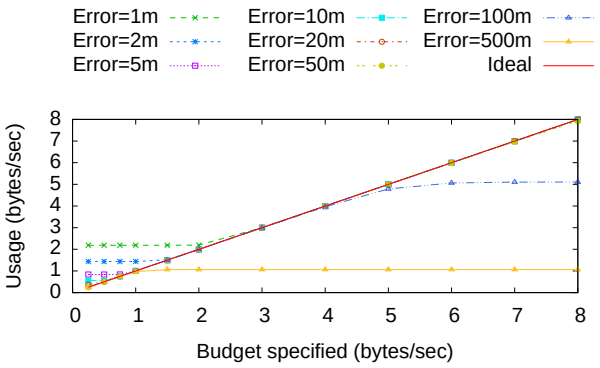

Fig. 16. Conformance with the specified budget, for delay optimizing sampler with various maximum error bounds.

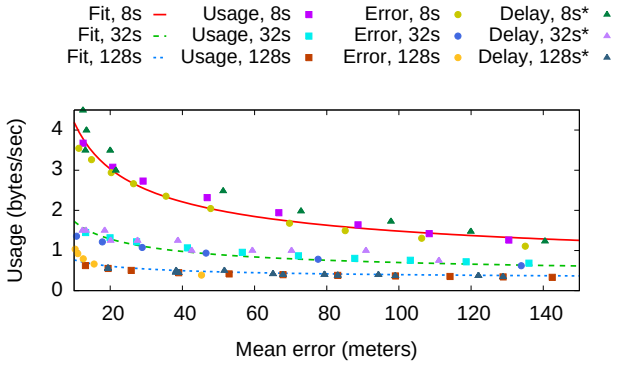

Fig. 17. Convergence of usage (μ), error (δ), and delay (τ) optimizing samplers along with conformance to solution of Equation 1. (*Approximate matching.)

we can get by with a smaller delay.

Finally, Fig 16 illustrates budget conformance. Practical constraints on delay, as well as the constant in Eq. 4, limit the adaptive power of the delay optimizing sampler. Specifically, it is possible to specify budget and error bounds so low that it is not possible to achieve both. In this case, the sampler will go over budget, as seen at the left part of Fig 16. For example, a budget below 2 bytes/sec is inadequate to enforce 1 meter maximum error bound, independent of delay. The sampler could potentially reject these settings, based on the expected uplink usage from Fig 9. However, in our current experiments, we simply record and report the budget conformance violation. Similarly, the sampler will not conform to an overly generous budget, if additional uplink usage would not result in improved accuracy or timeliness. However, as Fig 16 shows, for a reasonable error bound and given budget, we closely conform to the given budget while enforcing the error bound.

7.6 Sampler and Trade-Off Convergence

Above, we present a unified sampling framework with three samplers optimizing *uplink usage* (usage minimizing sampler in §7.3), *accuracy* (error minimizing sampler in §7.4), and *timeliness* (delay minimizing sampler at §7.5).

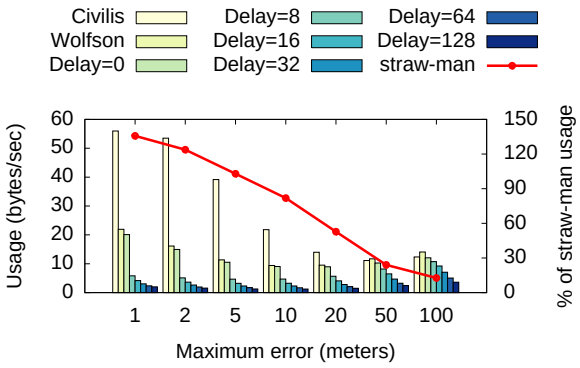

Fig. 18. Absolute usage with increasing maximum error bound for straw-man sampler (line graph), and percentage usage relative to straw-man for Čivilis, Wolfson, and unified extrapolator with various configured delays (bars).

Fig 17 supports our claim that Eq. 1 governs the relationship between these three aspects of sampler performance. Here, we plot the incurred error, uplink usage and delay of each of the three samplers (represented by points), together with the outcome predicted by Eq. 1 (represented by lines and indicated by Fit). We find close agreement between the samplers themselves, and between the samplers and the trade-off equation.

8 End-to-end Evaluation

In this section, we look at the end-to-end performance of our system, using our unified extrapolator and sampling framework together. Here, we focus on three basic scenarios: a user with a set uplink budget, a user with a set accuracy constraint, and a user with both accuracy and budget constraints, reported separately below. Where applicable, the *period* used below is 1 hour.

8.1 Tracking with Maximum Error and Delay

In our first scenario, the system operator specifies a maximum error and delay. Fig 18 shows the reduction of mean data usage of our end-to-end system, compared to straw-man, for several maximum error and delay settings. Here, the plotted line shows the absolute data usage (left axis) by a straw-man that samples as soon as the distance from the last sample exceeds X meters, where X is the configured error limit. Naturally, the straw-man uplink usage decreases as error tolerance grows. The bar graphs show the usage of other approaches, as a percentage of the straw-man's usage.

We also compare against two approaches from the literature, labeled Čivilis and Wolfson. Specifically, Čivilis represents our implementation of the *DSC modified segment-based policy* described in [23], by combining our map-based (M1) extrapolator with our usage minimizing sampler (with zero delay). Similarly, Wolfson represents our implementation of the *speed dead-reckoning* technique described in [21], by combining our constant velocity extrapolator with our usage minimizing sampler (also with zero delay).

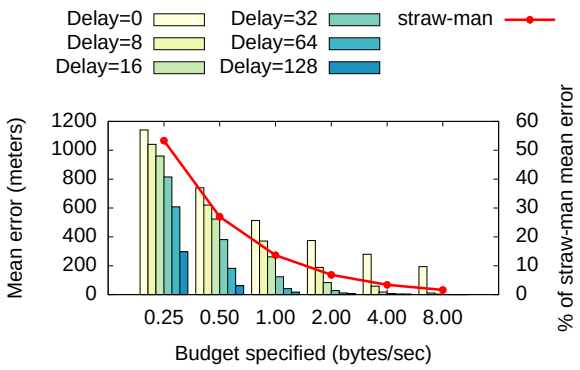

Fig. 19. Absolute mean error with increasing specified budget for straw-man sampler (line graph), and percentage mean error relative to straw-man for unified extrapolator with various configured delays (bar graphs).

Focusing on the case of 10 m maximum error bound, our system achieves a 77% reduction in data usage vs. the straw-man, with no delay. If the operator is willing to accept some delay, our system reduces data usage relative to the straw-man by 94% (17×) for 32 seconds delay and 96% (25×) for 64 seconds delay. Among the previous work, Čivilis' map-based solution suffers greatly for error bounds 20 m and below, due to the effects discussed in §6. Wolfson's solution, meanwhile, is competitive with ours with no delay configured.

Dataset	0s	8s	16s	32s	64s	128s
OSM	77%	88%	92%	94%	96%	97%
UIC	76%	87%	91%	93%	95%	96%
MSMLS	15%	16%	45%	67%	74%	81%

TABLE 3
Data usage reduction compared to straw-man for usage minimizing sampler with various delays (error=10 m).

Finally, Table 3 shows the end-to-end performance improvement of our system compared to the straw-man for all three of our datasets (shown in Table 2).

8.2 Tracking on a Budget and with Delay

In our second scenario, the system operator specifies an uplink budget and a delay. Fig 19 shows the reduction of mean error for our end-to-end system, relative to a straw-man solution that samples at a fixed time interval. Here, the straw-man period is selected to meet the specified budget. This straw-man essentially describes the state of the practice today (see §2).

Focusing on the case of 2 bytes/sec specified budget, we see that our system reduces mean error by 81% compared to the straw-man solution, with no additional delay. If the operator is willing to accept some delay, our system reduces mean error relative to the straw-man by 98% (50×) for 32 seconds delay, and 99% (100×) for 64 seconds delay.

Finally, Table 4 shows the end-to-end performance improvement of our system compared to the straw-man for all three of our datasets (shown in Table 2).

Dataset	0s	8s	16s	32s	64s	128s
OSM	81%	91%	96%	98%	99%	100%
UIC	70%	87%	94%	98%	99%	99%
MSMLS	92%	93%	97%	100%	100%	100%

TABLE 4
Mean error reduction vs. straw-man for error minimizing sampler with various delays (budget=2 Bytes/s).

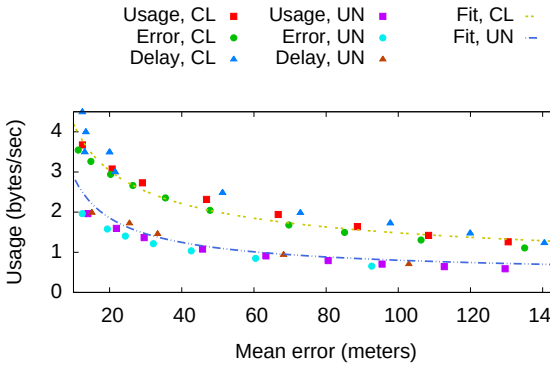

Fig. 20. Convergence and conformance for Constant Location (CL) and Unified (UN) extrapolators (delay=8s).

8.3 Tracking on a Budget, with an Error Limit

Finally, in our third scenario, the operator specifies a maximum error bound and a target budget. Here, no simple straw-man solution exists to compare against. Moreover, since the trade-off is the same regardless of which two constraints our user configures, performance as such does not differ between our sampling techniques. Thus, our interest here is largely in verifying that the delay minimizing sampler converges to the same performance as the error and budget minimizing samplers, and that all three conform to the trade-off in Eq. 1.

Fig 20 plots the performance of all three sampling techniques as points, for delays at 8 seconds. We show these results when using the unified extrapolator, as well as with the constant location extrapolator. Here, the exponents (B and C) in Eq. 1 are identical, but the coefficients (A and D) differ between extrapolators, due to the difference in extrapolator performance. The results clearly reinforce the trade-off relationship, and clearly demonstrate the performance improvement when using the more sophisticated extrapolator. Results are similar for other choices of delay.

9 Conclusion

Given the rising popularity of GPS tracking of people, vehicles and other possessions, reducing the uplink usage and providing performance controllability for such applications are pressing needs. Based on how data usage is calculated, we find that reducing the number of messages is the only effective means of reducing the uplink usage of a GPS tracking application. With this

in mind, we designed a unified *thrifty tracking* system, that predicts future movements, and adaptively samples the GPS trace to meet user-specified performance targets. Our system provides predictable tracking performance in terms of uplink usage, error, and delay, which combined with its considerable savings in terms of data usage, makes thrifty tracking an attractive solution for modern GPS tracking systems.

10 Acknowledgments

This material is based upon work supported by the U.S. National Science Foundation under Grants CNS-1017877, CNS-1149989 and DGE-0549489.

References

[1] J. Biagioni, A. Musa, and J. Eriksson, "Thrifty tracking: online gps tracking with low data uplink usage," in *SIGSPATIAL GIS*. ACM, 2013, pp. 486–489.
[2] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *MobiSys*. ACM, 2010, pp. 299–314.
[3] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao, "Energy-accuracy aware localization for mobile devices," in *MobiSys*. ACM, 2010.
[4] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *MobiSys*. ACM, 2010, pp. 315–330.
[5] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones," in *SenSys*. ACM, 2009, pp. 85–98.
[6] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær, "Entracked: energy-efficient robust position tracking for mobile devices," in *MobiSys*. ACM, 2009, pp. 221–234.
[7] "ThriftyTracker," http://www.cs.uic.edu/Bits/ThriftyTracker.
[8] D. H. Kim, Y. Kim, D. Estrin, and M. B. Srivastava, "Sensloc: sensing everyday places and paths using less energy," in *SenSys*. ACM, 2010, pp. 43–56.
[9] D. Ashbrook and T. Starner, "Using gps to learn significant locations and predict movement across multiple users," *Personal and Ubiquitous Computing*, vol. 7, no. 5, pp. 275–286, 2003.
[10] R. Jurdak, P. Corke, D. Dharman, and G. Salagnac, "Adaptive gps duty cycling and radio ranging for energy efficient localization," in *SenSys*. ACM, 2010, pp. 57–70.
[11] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica*, vol. 10, no. 2, pp. 112–122, 1973.
[12] J. Hershberger and J. Snoeyink, "An O(n log n) implementation of the Douglas-Peucker algorithm for line simplification," in *Proc. of 10th Annual Symposium on Computational Geometry*. ACM, 1994, pp. 383–384.
[13] G. Kellaris, N. Pelekis, and Y. Theodoridis, "Trajectory compression under network constraints," in *Adv. in Spatial and Temporal Databases*. Springer, 2009, pp. 392–398.
[14] N. Meratnia and A. Rolf, "Spatiotemporal compression techniques for moving point objects," in *EDBT*. Springer, 2004, pp. 765–782.
[15] T. H. N. Vu, K. H. Ryu, and N. Park, "A method for predicting future location of mobile user for location-based services system," *Computers & Industrial Engineering*, vol. 57, no. 1, pp. 91–105, 2009.
[16] Y. Ohsawa, K. Fujino, H. Htoo, A. T. Hlaing, and N. Sonehara, "Real-time monitoring of moving objects using frequently used routes," in *Database Systems for Advanced Applications*. Springer, 2011, pp. 119–133.
[17] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. Ravi, "Squish: an online approach for gps trajectory compression," in *Proc. of 2nd Int'l Conf. on Computing for Geospatial Research & Applications*. ACM, 2011, p. 13.
[18] A. P. Sistla, S. Dao, O. Wolfson, and S. Chamberlain, "Modeling and querying moving objects," in *ICDE*. IEEE, 1997, pp. 422–422.
[19] O. Wolfson, L. Jiang, A. P. Sistla, M. Deng, S. Chamberlain, and N. Rishe, "Databases for tracking mobile units in real time," in *ICDT*. Springer, 1999, pp. 169–186.
[20] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez, "Cost and imprecision in modeling the position of moving objects," in *ICDE*. IEEE, 1998, pp. 588–596.
[21] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha, "Updating and querying databases that track mobile units," in *Mobile Data Management and Applications*. Springer, 1999, pp. 3–33.
[22] C. S. Jensen and S. Pakalnis, "Trax: real-world tracking of moving objects," in *PVLDB*. VLDB Endowment, 2007, pp. 1362–1365.
[23] A. Civilis, C. S. Jensen, and S. Pakalnis, "Techniques for efficient road-network-based tracking of moving objects," *TKDE*, vol. 17, no. 5, pp. 698–712, 2005.
[24] A. Civilis, C. S. Jensen, J. Nenortaite, and S. Pakalnis, "Efficient tracking of moving objects with precision guarantees," in *Mobiquitous*. IEEE, 2004, pp. 164–173.
[25] R. Lange, F. Dürr, and K. Rothermel, "Online trajectory data reduction using connection-preserving dead reckoning," in *Mobiquitous*. ICST, 2008, p. 52.
[26] R. Lange, T. Farrell, F. Durr, and K. Rothermel, "Remote real-time trajectory simplification," in *PerCom*. IEEE, 2009, pp. 1–10.
[27] R. Lange, F. Dürr, and K. Rothermel, "Efficient real-time trajectory tracking," *VLDB Journal*, vol. 20, no. 5, pp. 671–694, 2011.
[28] "OpenStreetMap," http://www.openstreetmap.org.
[29] J. Biagioni and J. Krumm, "Days of our lives: Assessing day similarity from location traces," in *UMAP*. Springer, 2013, pp. 89–101.
[30] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *SIGSPATIAL GIS*. ACM, 2009, pp. 336–343.
[31] J. Krumm, "A markov model for driver turn prediction," *SAE SP*, vol. 2193, no. 1, 2008.
[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[33] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

A.B.M. Musa is a Ph.D. candidate in Computer Science at University of Illinois at Chicago (UIC). He received his B.S. in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET) in 2008. His research interests include networked systems, wireless networks, mobile sensing, and localization.

James Biagioni completed his Ph.D. at the University of Illinois at Chicago (UIC) in 2014, and received his B.S. from UIC in 2006. James' research interests are centered around the problem of inferring interesting and useful phenomena from large collections of sensor data.

Jakob Eriksson has been an Assistant Professor of Computer Science at the University of Illinois at Chicago since 2009. Jakob completed his Ph.D. at the University of California, Riverside, and his undergraduate degree at the Royal Institute of Technology (KTH), Sweden. Prior to joining UIC, he was a postdoctoral research associate at the Massachusetts Institute of Technology. His primary research interests are in computer systems and mobile sensing.